Вне области применения

Апр 26, 2013 — от автора admin

Мое становление как программиста началось в 2005 году и продолжается по сей день. Несомненно, многие читатели смогут похвастаться более обширным опытом, но речь пойдет о другом. Мой профессиональный рост наложился на интересный период - повышение культуры программирования в рунете, если это можно так назвать. Профессионалы были всегда, но сегодня подкованность среднего программиста(во всяком случае в сфере best pracices) несравнимо выше, чем тогда. И само по себе это не плохо, опасения вызывает сам тренд который привел к подобному результату. При его продолжении мы можем всерьез столкнуться с той же проблемой, с которой все и началось - а именно с говнокодом, но на сей раз говнокодом облагороженным, прикрытым множеством абстракций, а порой самими этими абстракциями и являющимся. Да-да, сегодня опять критикуем оверинжениринг.

Концепция паттернов не нова, идея о неизобретении своего велосипеда набила оскомину, за отстуствие тестов принято посыпать голову пеплом и каяться как на страшном суде. Достаточно прочитать примерно 5 книг по проектированию, чтобы понять, что сейчас модно и как нужно писать код. Загвоздка в другом - даже прочитав 25 книг вы не поймете когда нужно писать какой код. В какой-то мере можно научиться этому глядя на старших товарищей, которые выдают идеи того, когда можно применить то или иное решение, при этом ход мыслей, который привел к такому решению опять остается за кадром. В итоге у новичка в лучшем случае в голове получается некая карта, описывающая когда можно применять что-либо, причем карта довольно ограниченная. И горе новичку, если он

окажется впечатлительным, а результат будет отличным - у человека может появиться любимый паттерн. И он этот паттерн может бережно пронести скозь годы, лепя его в самых причудливых местах, и вызывая тем самым искреннее удивление и неподдельную ненависть коллег.

Краткий экскурс в права меньшинств

Вы не любите TDD? Любите писать код в процедурном стиле? Не используете EJB? Ай-ай-ай, да как у вас только земля под ногами не горит. Любой социум делится на группы, и если вы не готовы вступить в одну группку, то вас с радостью причислят к другой. Как и везде, где в основе логики поведения лежат биологические закономерности, между группами наблюдается конкуренция. Какая-то из групп выигрывает почетный приз Права Обладания Абсолютной Истиной, остальные же объявляются врагами рода человеческого, в лучше случае забавными маргиналами. Все, в общем-то понимают это, достаточно глянуть любой пост на хабре, посвященный оверинженирингу - он будет заплюсован, но в комментариях можно будет увидеть с десяток килобайт демагогии на тему почему все-таки автор неправ. Но сделать ничего нельзя - каждый кто захочет пойти против сложившегося порядка вещей, будет либо проигнорирован, либо осмеян. Ситуацию усугубляет то, что те авторитеты, которые могли бы подкрепить протест своим именем, как правило, в этом не заинтересованы - они уже сидят на непыльных должностях, где правила Совершенного кода в принципе работают, и им уже нет дела до того, что в соседнем сегменте софтостроения люди ломают копья в спорах на тему "как же правильно писать код". Пока не произойдет перелома тренда с "писать совершенный код" на "писать код, решающий текущую задачу", мы обречены лихо писать на собеседованиях тестовый код с реализацией паттерна, которые нам не приходило в голову использовать даже в самом страшном сне, а потом, после

устройства на работу, не менее лихо штамповать тесты, тестирующие ничто, чтобы добиться заветных 100% покрытия кода. Самое забавное, что находятся люди, готовые искренне и горячо спорить на тему, какой же на самом деле должна быть реализация паттерна под тем или иным названием, опуская тот факт, что имя по поводу которого они спорят - всего лишь условность для человека, решившего записать свой личный опыт именно под таким названием. Самое печальное, что находятся и те, кто слушает предыдущих и, горестно качая головой, тут же каются, что дескать не смогли они правильно понять определение, обещая стать лучше. Но здоровую самокритику невозможно заменить нездоровым самобичеванием, и лучше они не становятся.

На вершине Олимпа

Мартин Фаулер - наш царь, отец и бог. Все мы чтим его как человека, надававшего линейкой по рукам несметным полчищам начинающих программистов, и во многом определившего мировоззрение целого поколения ИТ-шников. Мы скорее всего никогда не узнаем как и почему он стал мерилом хорошести кода. Быть может его книги стали той самой заглушкой для потока некачественных программистов, вливавшихся в отрасль до момента пока пузырь доткомов не лопнул. А может быть он правда первым, кто систематизировал все то, что было накоплено в плане проектирования программ к тому времени. Все это дела давно минувших дней. А сегодня мы имеем набор рекомендаций, ставших негласными правилами хорошего тона в разработке. Возможно эти правила и хорошие, попробуем очертить границы их применения: 1) Языки со статической типизацией - в первую очередь C++ и Java. 2) Крупные программные продукты, создающиеся из расчета взаимозаменяемости программистов, участвующих в проекте. 3) Программные продукты, рассчитанные на дальнейшее развитие и длительное сопровождение. 4) ПО рассчитанное на довольно жесткие требования надежности - корпоративные решения

в общем. Вроде бы нормальные условия, покрывающие немалую долю рынка ПО. Проблема в том, что остальная доля рынка освещена в подобных талмудах... никак. Да, она просто игнорируется, вернее будет правильней сказать, что подходы описываемые в книгах просто не подходят для остальных продуктов.

Черный рынок

А между тем, если присмотреться, то можно заметить колоссальное число мелких и средних проектов, которые не подходят под условия, описанные выше. Тут и доминирование скриптовых языков, и команды с тимлидом, которого так просто не выкинешь, потому что остальные джуниоры набирались в угоду минимальному бюджету, а то и вообще работающие за идею и призрачные опционы, обещающие быть хорошей прибавкой у стипендии, и невнятные прогнозы касательно развития проекта. Немалое количество отраслевого софта не меняется годами - программы просто обслуживают техпроцессы, для которых были созданы, а изменения, вносимые в них - чисто косметические. Конечно, код находящийся там чаще всего оставляет желать лучшего, но он вполне расширяем в рамках тех требований, которые к нему ставятся. Еще более изящной неопределенность будущего ПО может выступать в случае, когда будет ли оно развиваться определяется тем, как быстро оно сможет выйти на рынок и отбить вложения, чтобы получить средства для дальнейшего развития. Абстракции не всегда значительно увеличивают объем кода, но всегда увеличивают время его написания. Наверняка есть игрушечные(получившие со старта огромное финансирование) и не в меру удачливые стартапы, которые могут себе позволить такую роскошь как написание по всем параметрам богоугодного кода. И это здорово - должны же быть где-то на земле счастливые программисты. Но не следует рассматривать такие проекты как закономерность. Отказоустойчивость - скорее вопрос предпочтений. Мы понимаем, что наши программы должны

работать стабильно. Вопрос в том, насколько стабильно . Все что пишется нами, пишется для решения какой-то задачи. И если в постановке этой задачи не требуется обеспечить 100% отказоустойчивость, то скорее всего она и правда не нужна. Все имеет цену - за отказоустойчивость надо платить, тратить время на создание оберток, проверок, систем резервирования. И в конечном итоге наша задача не создать совершенно неубиваемую систему, а сделать так, чтобы затраты на повышение надежности системы оказались меньше затрат вызванных ее простоем. Звучит кощунственно, но при трезвом рассмотрении так оно и есть. Многие небольшие проекты интересны тем, что для них важнее быть сданными в сжатые сроки, чем соответствовать требованиям качества. Да, это печально, да это неправильно, да это может привести к печальным последствиям в будущем, но такая модель жизнеспособна и распространена. Достопочтенные мэтры программирования, потирающие руки в предвкушении работы по переписыванию этого говнокода к нормальному виду, отказываются отдавать себе отчет в том, что если бы не этот говнокод, на котором удалось запустить проект, то работы для них просто не было бы. У них нет понимания того, что те деньги, которые заказчик готов платить им сейчас , появились в результате худо бедно работающего куска говнокода. Кстати именно с этого сегмента рынка начинают свой профессиональный путь почти все программисты. И им на голову сыпятся советы, один замечательнее другого. Но реальность такова, что следуя этим советам на данном этапе своего развития и при участии в таких проектах - это прямая дорога к провалу. Парадокс ситуации в том, что те, кто может реально следовать хорошим практикам, в данном сегменте рынка либо не работают, либо вообще не нужны из-за высоких расценок. А те, кто не может пока эффективно исполнять роль "хорошего программиста", наоборот всячески присутствуют на рынке труда в данном сегменте и активно шпыняются на предмет невыполнения заветов совершенного кода. Нельзя сказать, что новичкам советуют использовать паттерны, тестирование, расширяемую архитектуру по злому умыслу, нет. Все

эти советчики искренне желают им добра, не замечая при этом, что задачи решаемые ими несколько отличаются. Применение хороших практик тут не даст положительной обратной связи, они просто не подходят для решения поставленных тут задач. Варясь в котле мелких и средних проектов и пытаясь писать совершенный код, новичок обречен снова и снова срывать сроки и делать то, что никому не нужно. Да чего греха таить, ваш покорный слуга в свое время с честью завалил 2 многомесячных проекта на фрилансе, превысив все мыслимые сроки, а также с позором был выперт с двух работ по причине низкой продуктивности. Причина проста - я пытался делать все по уму.

Мракобесие отраслевых масштабов

Всякий начинающий программист сталкивается с дилеммой двух стульев. Обучение, согласно законам логики нашей вселенной, возможно двумя путями - принятием информации, необходимой для достижения цели, на веру, либо же проверкой всех возможных вариантов достижения цели с поиском подходящего. И оба варианта оказываются не то что бы идеальными. Усвоение имеющейся информации - хороший способ, если мы знаем, что она достоверна. И она достоверна, она чертовски достоверна, я готов согласиться, что авторы книжек по проектированию использовали все даваемые им советы в своих проектах. Я только не верю, что нашему программисту попадется структурно идентичный проект, а также, что начинающий программист будет допущен к работе над структурно идентичным проектом, во всяком случае на том уровне, где он будет решать какие подходы ему применять... Кстати, как проверить достоверность информации, даваемой в таких книгах? Только ударившись в область допущений, что код будет меняться так или вот так. Любой второкурсник сможет к примерам, описываемым в книгах привести контр-примеры, которые будут лаконичнее и эффективнее, решая при этом ту же задачу.

Любую хорошую практику практически невозможно фальсифицировать, что заставляет задуматься о принятии ее как непогрешимой истины, дающей ответ на вопрос о том, как писать программы. Эти практики в определенных случаях повышают вероятность того, что в случае возможного расширения кода вы избежите некоторых проблем. Не более. Второй вариант - это самостоятельный поиск решения. Отношение к таким людям мягко говоря негативное, а чаще всего - презрение. Негатив идет по двум причинам: во-первых, сформировано общественное мнение, гласящее, что человек не использующий best practices - это и не человек вовсе, во-вторых, обучающийся на своем опыте, неизбежно будет совершать ошибки. Как правило, ошибки эти подмечают его более опытные коллеги, для которых подобные огрехи - и правда детский лепет. Беда в том, что формализация процесса обучения программированию пока отсутствует, причем даже в головах тех, кто через этот процесс прошел. Например, вы состоявшийся программист, попробуйте во время проектирования очередного модуля или сколько-то серьезного класса задавать вопрос "почему" на каждое свое действие, попробуйте с точки зрения логики обосновать последовательность ваших действий, так чтобы одно однозначно следовало из другого. Чтобы сэкономить ваше время, скажу, что у вас не получится. Если получится, то ваша твердолобость достойна всяких похвал, этот же пост прочтения вами не достоин, закройте его. В процессе создания архитектуры вы оперируете не знаниями, а скорее опытом. Вы не сможете прыгнуть выше своей головы, создав архитектуру по книжке, до тех пор пока не придете к необходимости подобных построений сами. Но можете обмануть интервьюера на собеседовании, использовав обратное утверждение.

Эвристическое бессилие

В волшебной стране молочных рек и розовых единорогов,

умирающих от почечной недостаточности ввиду отсутствия другой влаги, люди всегда способны выводить из общего частное. К сожалению, их постигает та же судьба, что и единорогов, поэтому добраться до нашего с вами мира они не успевают, чтобы привнести в него это сакральное знание. Поэтому практически все люди не способны на основании знания чего-то общего, абстрактного, вывести частное, кроме тех случаев когда речь идет о простых и однозначных вещах. Проектирование ПО такой не является. Я тщетно и долго искал людей, которые были бы способны прочитать банду четырех, а потом строить абстракции и успешно применять их в своем коде. Если что, речь о людях без опыта. Если что, синглетон не в счет. Что-то в голове не позволяет брать и плодить штучные абстракции по образу и подобию прочитанного: либо человек не понимает зачем это делается и в итоге абстракция создает проблем больше, чем решает, либо он честно признается себе, что не видит куда это можно приткнуть. Возможно есть те, кто способен на такой способ обучения, но их мне встречать не доводилось. А вам? По моим наблюдениям, вывод общего из частного - гораздо более посильная задача, с которой справляются все. Да, я про тот случай, когда человек сам приходит к паттернам побившись о грабли. И который открывает книжку только затем, чтобы понять как общепринято называется то, что он узнал на своем опыте. Если он так ничего и не узнал к моменту открытия книжки, то тут уже не сработает никакой подход. Знаний на уровне формального описания предмета часто недостаточно для успешного его применения. Они должны быть встроены в логические цепочки на уровне фактов, которые должны быть не просто заучены, но прочувствованы. Обычно такое осмысление приходит в процессе практики(реже - когда, через несколько месяцев или лет вы чувствуете, что что-то в голове встало на свои места и вы готовы этим пользоваться). Причем независимо от того читал ли программист про паттерны или нет. Если задачи будут показывать неэффективность его текущего подхода, то он придет к решению, которое будет работать лучше. Неважно как он обзовет это у себя в голове. Объяснить получившийся паттерн другому

программисту - минутное дело. И к тому же, если описание паттерна формируется в мозгу каждого программиста на тех примерах с которыми он имел дело, то, даже имея общее определение, два человека, используя его, будут говорить о довольно разных вещах, что опять таки делает привязку к официальным паттернам нежелательной. Я даже возьму на себя наглость сказать, что если программисты постоянно используют в работе официальные названия паттернов, и более того ограничиваются только этими названиями, не обсуждая логику с ними связанную, то скорее всего это либо вчерашние студенты, либо заядлые любители Java, где использование паттернов для обхода ограничений языка суть рутина.

Объять необъятное

Все практики нацеленные на создание расширяемого кода уходят из предметной области в область предположений, что этот код будет поддерживаться и расширяться. И в этом заключается величайшее отличие программирования от любой другой инженерной деятельности - обычно, если агрегату не хватает мощности или функций, то его заменяют на другой агрегат, более подходящий. Негласный принцип программирования гласит, что вместо того, чтобы создавать заново, лучше расширить или изменить уже имеющееся. Благо технология позволяет. К слову технология позволяет и апгрейдить физические устройства для достижения ими нужных функций. Тем не менее это почему-то не делается. Наверняка из соображений чисто экономического характера. Имея два подхода - переписывание с нуля и расширение готового мы привыкли к тому, что второй вариант лучше, менее трудозатратнее. При этом забывая, что: а) в случае криво написанного нерасширяемого кода мы получаем большие сложности с добавлением функционала б) в случае создания расширяемого кода стоимость переписывания закладывается уже на этапе первоначального написания кода

Трудозатраты будут в обоих случаях, вопрос лишь в том, когда они больше. И именно на этот вопрос по-хорошему должен знать ответ опытный программист. Опытный архитектор может дать задание написать говнокод, но с одним условием - никто не должен про это узнать.

Область применения

Вы можете годами не находить себе места в связи с тем, что не можете найти места практикам, о которых вы читаете, но это скорее всего не значит, что с вами что-то не так. Неважно сколько паттернов вы знаете и используете, важно лишь то, что вы можете применить их к месту. Я вообще убежден что объем знаний хорошего проектировщика можно освоить за пару месяцев, но чтобы научиться правильно их применять потребуются годы. В случае если вы не работаете над системой, содержащей минимум десятки сильно связанных классов(а таких систем очень много), вам не нужно забивать голову как лучше учинить оргию из разнообразных абстракций так, чтобы потом все это выглядело как легкая эротика. Вы вообще можете быть хорошим программистом и ни разу не вляпаться в такой проект, путешествуя между маленькими проектами и системами с низкой связанностью. Если вы все таки хотите понять кто и зачем все это использует, то изучайте код больших проектов на Java, C#, C++. В особенности это будет полезно программистам на скриптовых языках, потому что так можно будет наглядно увидеть ту разницу в подходе к программированию на, например, Java и Python. В любом случае, даже если вы найдете реализацию того или иного паттерна, не следует принимать это близко к сердцу, паттерны обычно рождаются или выбираются в рамках конкретного проекта. Так что велика вероятность того, что увиденное там не может быть применено в текущем вашем проекте. Если начальство требует от вас писать "правильный код" - пишите! Вы либо правда еще не понимаете, что

с вашим профессиональным уровнем не так, и тогда вам полезно будет сделать так, как говорят, если же ваш тимлид человек идейный, вот и не спорьте с ним - рядовому программистишке его веру не разрушить, а пинки потом получать не вам.

Что делать?

Думать, хорошо думать прежде чем использовать что-то что вы не понимаете. Если оно кажется вам ненужным, то оно и правда может таким быть. Если вы видите, что тесты нужны вашей программе как собаке боковой карман - не пишите их, вы все равно не сможете написать их так, чтобы они приносили пользу. Если вы ознакомились со списком паттернов, а потом с ужасом обнаружили, что в вашем коде за последние пару лет в лучшем случае нашли пристанище 3-4 из них, но в остальном вы довольны своим кодом, и с его сопровождением нет проблем, то оставьте все как есть. И, конечно, помните, что способности людей к агрессивной риторике всегда обратно пропорциональны их доказательной базе. И каждый раз когда вам будут в качестве аргумента к использованию какого-то подхода аппелировать к гипотетической ситуации, вероятность возникновения которой подозрительно мала, просто вспомните при каких условиях бабушка может стать дедушкой.

ссылка на оригинал статьи http://habrahabr.ru/post/178035/